Fuentes Perez, Jhon Angel

Ortega Caicedo, Jhonatan Andres

Torres Valencia, Daniel

Kata De Arquitectura

Docente: Julio Robles.



Universidad de San Buenaventura

Facultad de Ingenieria (Cali)

Ingeniería de Sistemas

Santiago de Cali, Colombia

2025

Índice

[**Contexto 3**](#_ai4wrv1og6zd)

[**Lista De Interesados 3**](#_erv6wormbrp0)

[**Requisitos Funcionales 4**](#_ywrujcqcqv72)

[**Requisitos No Funcionales 5**](#_gwb6ufdewr3b)

[**Atributos de Calidad por Interesado 6**](#_b9o5x9mkn5zy)

[**Ponderación 7**](#_xx2jew3c9c8i)

[**Drivers Arquitectónicos 8**](#_ut10c7m6qwfz)

[**Definir Tácticas para atacar los atributos de calidad 8**](#_iwwz4h7kys4x)

[**Patrones Arquitectonicos 11**](#_mnbl38d7mg1p)

[**Definir la Estrategia (Microservicios + API Gateway) 11**](#_27ku34joq1i6)

[**Modelo 4 + 1 vistas 14**](#_6xmxze4iym0w)

[Vista Lógica 14](#_bqhjcqvnfgik)

[Vista Desarrollo 17](#_hbzfyvcez4lq)

[Vista Procesos 20](#_fsptcr46g3at)

[Vista Física 23](#_qepqfhl2uwyz)

[Vista De Escenarios 26](#_foozggswdgft)

# Contexto

El sistema Servidor de tiquetes surge como una solución intermediaria orientada a facilitar la distribución y venta de tiquetes de eventos a través de revendedores, proporcionando una API robusta, escalable y confiable. Su objetivo principal es ofrecer a los revendedores una plataforma centralizada para consultar la disponibilidad de tiquetes, realizar compras seguras sin riesgo de duplicidad, y recibir notificaciones en tiempo real sobre cambios de estado relevantes. Además, se contempla una interfaz administrativa para gestión interna, un sistema de comisiones por ventas, y una arquitectura capaz de adaptarse a regulaciones legales específicas por país. Esta plataforma busca también reducir la carga técnica de los revendedores al ofrecer opciones de hosting integradas, permitiendo que cada actor del ecosistema se enfoque en maximizar su alcance comercial y operatividad sin preocuparse por los aspectos técnicos de fondo.

# Lista De Interesados

**1. Revendedores de tiquetes**

Por qué: Son los principales usuarios del API. Necesitan consultar disponibilidad, vender tiquetes, recibir notificaciones sobre eventos relevantes (como fraudes) y obtener reportes financieros para su gestión comercial.

**2. Usuarios finales (compradores de tiquetes)**

Por qué: Aunque no interactúan directamente con el sistema central, su experiencia depende de la disponibilidad en tiempo real, la confiabilidad del proceso de compra y la protección contra errores como la doble compra.

**3. Equipo de desarrollo**

Por qué: Son los encargados de diseñar, construir y mantener el sistema. Necesitan entender los requerimientos funcionales y no funcionales, así como prever problemas de escalabilidad, concurrencia y cumplimiento legal.

**4. Equipo de soporte técnico**

Por qué: Utilizarán la interfaz administrativa para ayudar a revendedores con problemas técnicos o solicitudes especiales. Necesitan herramientas de monitoreo y control interno del sistema.

**5. Administradores del sistema**

Por qué: Gestionan las operaciones generales, políticas de comisiones, cumplimiento normativo y relaciones comerciales con los revendedores. Requieren visibilidad completa sobre el uso del sistema y herramientas para tomar decisiones estratégicas.

**6. Entidades regulatorias por país**

Por qué: Dependiendo del país, pueden existir leyes específicas sobre reventa de tiquetes, protección al consumidor, privacidad de datos, antifraude y comercio electrónico. El sistema debe cumplir con estas normativas, por lo que podrían requerir auditorías o segmentación de datos por jurisdicción.

**7. Equipo de seguridad y cumplimiento**

Por qué: Encargados de garantizar la integridad del sistema frente a fraudes, accesos indebidos o violaciones legales. Su trabajo es clave en la prevención de transacciones sospechosas o el manejo de eventos de riesgo.

# Requisitos Funcionales

| **ID** | **Descripción** | **Detalles** | **Interesados** |
| --- | --- | --- | --- |
| RF-01 | Consulta de tiquetes disponibles | El sistema debe permitir a los revendedores obtener la lista actualizada de tiquetes disponibles para un evento específico. | Revendedores, Equipo de desarrollo, Usuarios finales |
| RF-02 | Compra segura de tiquetes | El sistema debe garantizar que cada tiquete se pueda comprar una sola vez, incluso en escenarios de alta concurrencia. | Revendedores, Usuarios finales, Equipo de desarrollo, Seguridad |
| RF-03 | Notificaciones en tiempo real | El sistema debe enviar notificaciones a los usuarios cuando un tiquete que están observando ha sido comprado por otro. | Usuarios finales, Revendedores, Equipo de desarrollo |
| RF-04 | Panel administrativo interno | El sistema debe incluir una interfaz para que el administrador brinde soporte técnico, gestione eventos y consulte datos del sistema. | Administradores, Soporte técnico, Equipo de desarrollo |
| RF-05 | Generación de reportes financieros por revendedor | El sistema debe generar reportes detallados para que cada revendedor vea ingresos, ventas realizadas y comisiones acumuladas. | Revendedores, Administradores, Equipo de desarrollo, Soporte técnico |

# Requisitos No Funcionales

| **ID** | **Descripción** | **Detalles** | **Métricas** | **Interesados** |
| --- | --- | --- | --- | --- |
| RNF-01 | Alta disponibilidad | El sistema debe estar disponible en todo momento, especialmente durante eventos con alta demanda. | Uptime ≥ 99.9% mensual | Revendedores, Usuarios finales, Administradores, Infraestructura |
| RNF-02 | Escalabilidad | El sistema debe manejar desde miles hasta millones de usuarios simultáneamente sin degradación. | Soportar al menos 10,000 solicitudes por segundo sin pérdida de rendimiento | Equipo de desarrollo, Infraestructura, Administradores |
| RNF-03 | Seguridad y prevención de fraudes | El sistema debe prevenir compras duplicadas y detectar patrones de fraude en pagos. | 100% de prevención de doble compra, alertas de fraude ≤ 5 min posterior a la detección | Equipo de seguridad, Administradores, Revendedores |
| RNF-04 | Cumplimiento legal y segmentación por región | El sistema debe poder segmentar usuarios por país para cumplir con normativas específicas. | Soporte para separación de datos/reglas por al menos 5 jurisdicciones | Administradores, Equipo legal, Equipo de desarrollo |
| RNF-05 | Tiempo de respuesta del API | Las respuestas del API deben ser rápidas incluso en cargas altas. | Tiempo de respuesta promedio < 500 ms, 95 percentil < 1 s | Revendedores, Usuarios finales, Equipo de desarrollo, Infraestructura |

# Atributos de Calidad por Interesado

| **Interesado** | **Disponibilidad** | **Rendimiento** | **Seguridad** | **Compatibilidad** | **Escalabilidad** | **Total** |
| --- | --- | --- | --- | --- | --- | --- |
| **Revendedores** | 25% | 30% | 20% | 15% | 10% | 100% |
| **Usuarios finales** | 30% | 40% | 20% | 10% | 0% | 100% |
| **Administradores** | 20% | 10% | 30% | 30% | 10% | 100% |
| **Equipo de desarrollo** | 10% | 25% | 25% | 20% | 20% | 100% |
| **Equipo de seguridad** | 5% | 10% | 70% | 15% | 0% | 100% |
| **Infraestructura / DevOps** | 35% | 30% | 15% | 10% | 10% | 100% |
| **Total** | 20.83% | 24.17% | 30% | 16.67% | 8.33% | 100% |

# Ponderación

| **Atributo** | **SubCategoría** | **Descripción** | **Métrica** | **Impacto** | **Dificultad** | **% Peso** | **Valor** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Disponibilidad | Fiabilidad: Disponibilidad | Capacidad del sistema de estar operativo en momentos críticos. | ≥ 99.9% de tiempo en línea (uptime mensual) | 3 | 2 | 20.83 | 104.15 |
| Rendimiento | Eficiencia de desempeño: Tiempo | Capacidad del sistema para manejar el crecimiento de carga. | ≤ 300 ms bajo carga de hasta 1000 req/seg (escenario de estrés) | 2 | 3 | 24.17 | 121.35 |
| Seguridad | Seguridad: Protección | Capacidad para proteger información y prevenir accesos no autorizados. | 0 vulnerabilidades críticas detectadas en producción | 3 | 2 | 30 | 150 |
| Compatibilidad | Interoperabilidad | Capacidad de cumplir normativas específicas y adaptar funciones por región. | 100% cumplimiento de normativas regionales en países objetivo | 3 | 3 | 16.67 | 100.02 |
| Escalabilidad | Modularidad | Facilidad con la que el sistema puede ampliarse para atender mayor demanda. | Nuevos módulos integrables sin refactorización en >90% de los casos | 2 | 3 | 8.33 | 41.65 |

# Drivers Arquitectónicos

| **#** | **Atributo** | **Valor** |
| --- | --- | --- |
| 1 | Seguridad | 150 |
| 2 | Rendimiento | 121.35 |
| 3 | Disponibilidad | 104.15 |
| 4 | Compatibilidad | 100.02 |
| 5 | Escalabilidad | 41.65 |

# Definir Tácticas para atacar los atributos de calidad

**1. Seguridad**

| **Táctica** | **Descripción** |
| --- | --- |
| Autenticación fuerte | Uso de JWT para validar la identidad de revendedores y usuarios finales. |
| Autorización basada en roles | Control granular de acceso a recursos, separando funciones de administrador, revendedor y usuario. |
| Validación de datos | Sanitización estricta de entradas para prevenir inyecciones o corrupción de datos. |
| Detección y aislamiento de fraude | Sistema de reglas para monitorear patrones anómalos de compra y pagos. |
| Comunicación segura (TLS) | Encriptación de todas las comunicaciones cliente-servidor. |

### 

**2. Rendimiento**

| **Táctica** | **Descripción** |
| --- | --- |
| Cacheo de datos | Uso de Redis para almacenar temporalmente tiquetes disponibles o eventos frecuentes. |
| Asincronía en operaciones pesadas | Cola de mensajes con Amazon SQS para procesar tareas no críticas fuera del flujo principal. |
| Balanceo de carga | Distribución de solicitudes entre múltiples instancias para evitar saturación con Elastic Load Balancer (ELB). |

### 

**3. Disponibilidad**

| **Táctica** | **Descripción** |
| --- | --- |
| Failover automático | Mecanismos para redirigir tráfico ante falla de una instancia o zona por medio del API Gateway. |
| Monitoreo y alertas proactivas | Detección temprana de caídas o degradación con herramientas con Grafana. |
| Timeouts y reintentos controlados | Manejo robusto de fallas temporales con lógica de retry y circuit breaker. |

### 

**4. Compatibilidad**

| **Táctica** | **Descripción** |
| --- | --- |
| Segmentación de datos por región | Aislamiento físico o lógico de bases de datos/reglas por país (multi-tenant o multi-region). |
| Externalización de reglas legales | Motor de reglas (Rule Engine) parametrizable según regulación local. |
| Soporte multilingüe y multiformato | Adaptar el sistema a idiomas, monedas y formatos según región. |
| Registro y cumplimiento (compliance logging) | Logs específicos para normativas como GDPR, PCI-DSS o locales. |

**5. Escalabilidad**

| **Táctica** | **Descripción** |
| --- | --- |
| Escalado horizontal | Agregar más instancias del servicio según demanda, especialmente para APIs de consulta y compra. |
| Arquitectura basada en microservicios | Desacoplamiento de módulos (compra, catálogo, pagos, reportes) para escalar individualmente. |
| Event-driven architecture | Uso de eventos para desacoplar flujos (ej. compra → notificación, compra → comisión). |
| CDN para contenidos estáticos | Uso de redes de distribución para acelerar carga de imágenes, JS, etc. |

# Patrones Arquitectonicos

Para el diseño del sistema ‘Servidor de tiquetes’, la adopción de una arquitectura basada en Microservicios complementada con un API Gateway representa la solución más robusta y flexible frente a los desafíos clave del dominio: alta concurrencia, seguridad transaccional, separación regional/legal y escalabilidad.

La arquitectura de microservicios permite descomponer la plataforma en servicios independientes y altamente cohesionados, como compras, catálogo, pagos, comisiones, notificaciones y panel administrativo. Esta separación favorece el despliegue, escalado y mantenimiento individual de cada módulo sin afectar a los demás, lo que es fundamental en un entorno con miles de revendedores y millones de usuarios finales.

Por su parte, el API Gateway actúa como fachada única y segura hacia el sistema, controlando la autenticación, el enrutamiento, el versionamiento y la adaptación de servicios para los revendedores, permitiendo desacoplar el frontend o integraciones externas del backend distribuido. Además, habilita lógicas específicas por región o cliente, facilitando el cumplimiento normativo de jurisdicciones con reglas diferentes.

# Definir la Estrategia (Microservicios + API Gateway)

**1. Definición de dominios de negocio como microservicios**

| **Servicio** | **Funcionalidad Principal** |
| --- | --- |
| **Catalog Service** | Publicar, actualizar y consultar eventos y tiquetes disponibles. |
| **Purchase Service** | Validación, reserva y confirmación de compra; previene la doble venta. |
| **Payment Service** | Procesamiento de pagos, integración con PSPs, registro y detección de fraudes. |
| **Notification Service** | Envío de alertas en tiempo real (tiquete comprado, fraude, cambios de estado). |
| **Commission Service** | Cálculo y liquidación de comisiones por revendedor. |
| **Reporting Service** | Generación de informes financieros y operativos por revendedor. |
| **Administrative Service** | Funciones de soporte interno: desbloqueo, gestión manual de conflictos. |
| **Auth Service** | Gestión de autenticación/autorización de revendedores y usuarios finales. |

#### 

**2. Diseño e implementación del API Gateway**

* **Tecnología:** Amazon API Gateway.
* **Funciones clave del API Gateway:**
  + Autenticación y autorización con JWT.
  + Rate limiting y control de tráfico por revendedor.
  + Adaptación de respuestas por región.
  + Agregación de datos entre servicios para consultas compuestas (ej. catálogo + disponibilidad).

#### 

**3. Orquestación y Comunicación**

* **Síncrona:** REST entre servicios para acciones críticas (compra, consulta puntual).
* **Asíncrona:** AWS SNS/SQS para propagación de eventos como:
  + Compra completada → Notificación al usuario.
  + Pago detectado como fraudulento → Evento hacia soporte.
  + Evento modificado → Actualización en catálogo.

**4. Estrategias de despliegue y escalado**

* **Escalado independiente:** Cada microservicio escalará según su carga específica (ej. compras > comisiones).
* **Despliegue por región/jurisdicción:** Posibilidad de levantar instancias separadas del sistema por país, usando el gateway como punto de entrada adaptativo.

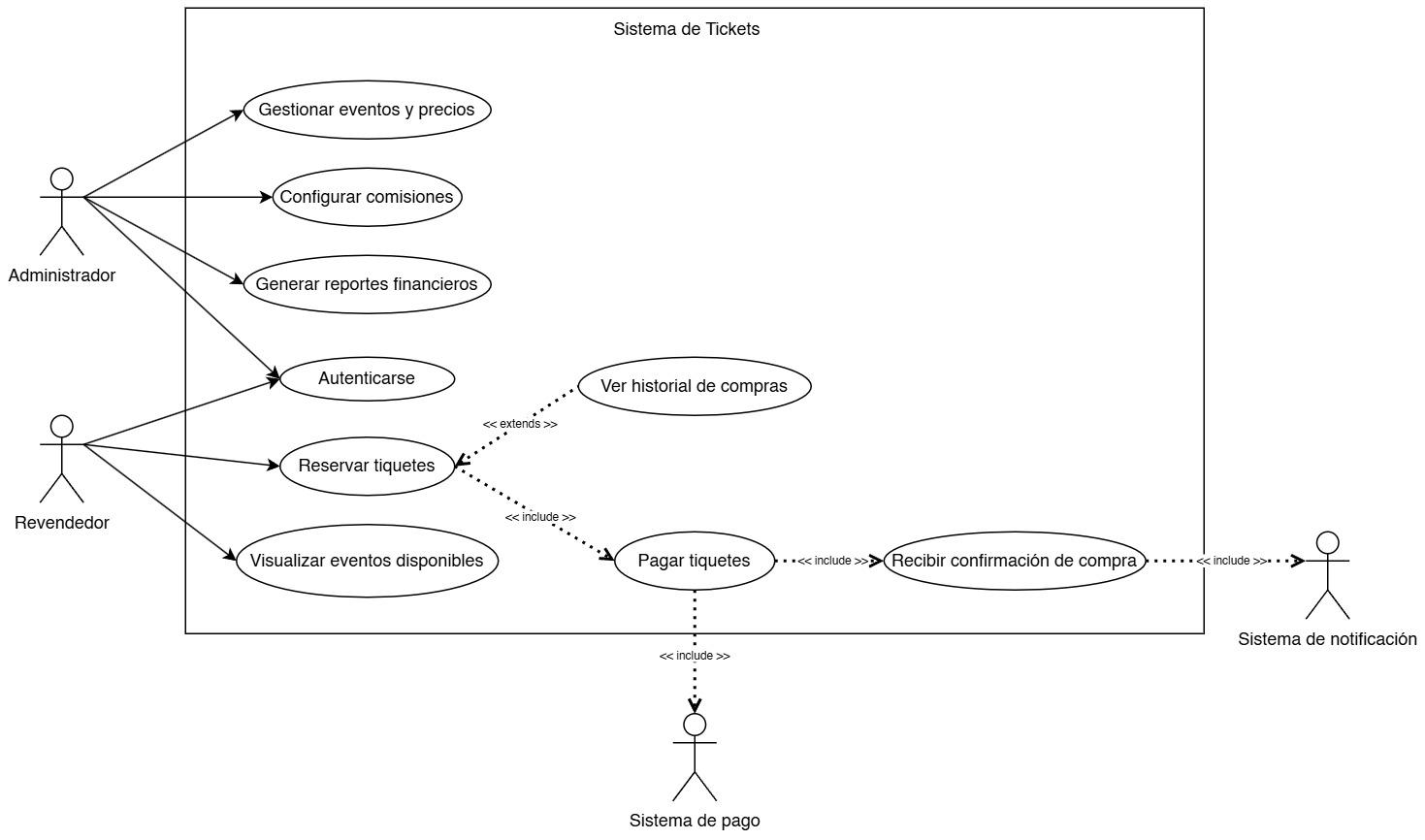
#### 

**5. Soporte para DevOps y Observabilidad**

* **Integraciones:**
  + **Service Mesh con AWS App Mesh:** para trazabilidad y políticas entre microservicios.
  + **Grafana y Amazon CloudWatch:** para métricas, logs y rastreo distribuido.

# Modelo 4 + 1 vistas

## Vista De Escenarios

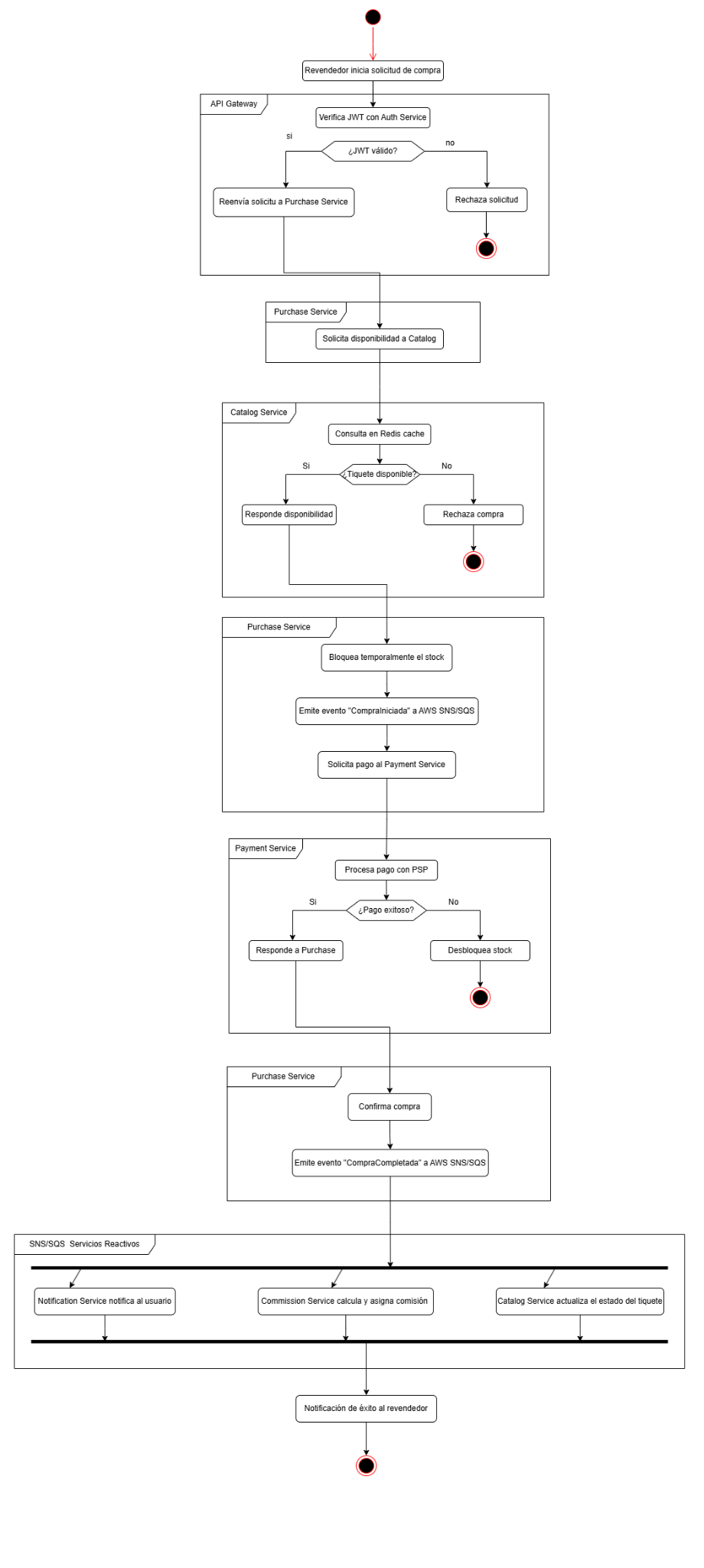


Este diagrama de casos de uso muestra las principales funcionalidades del Sistema de Tickets desde la perspectiva de los actores Administrador y Revendedor, así como las interacciones con sistemas externos: Sistema de Pago y Sistema de Notificación. El Administrador puede autenticarse, gestionar eventos y precios, configurar comisiones y generar reportes financieros. El Revendedor también debe autenticarse, puede visualizar eventos disponibles, reservar tiquetes (lo que incluye pagar y recibir confirmación) y consultar su historial de compras. El caso de uso “Reservar tiquetes” incluye operaciones obligatorias como “Pagar tiquetes” y “Recibir confirmación de compra”, y se extiende opcionalmente con “Ver historial de compras”. Además, el sistema se conecta con servicios externos: el sistema de pago para procesar la transacción y el sistema de notificación para confirmar la compra al usuario.

Tabla de elementos y relaciones

| **Actor / Sistema Externo** | **Elemento UML** | **Relación** |
| --- | --- | --- |
| Administrador | Autenticarse | Asociación directa |
| Administrador | Gestionar eventos y precios | Asociación directa |
| Administrador | Configurar comisiones | Asociación directa |
| Administrador | Generar reportes financieros | Asociación directa |
| Revendedor | Autenticarse | Asociación directa |
| Revendedor | Visualizar eventos disponibles | Asociación directa |
| Revendedor | Reservar tiquetes | Asociación directa |
| Revendedor | Ver historial de compras | Relación <<extends>> con Reservar tiquetes |
| Reservar tiquetes | Pagar tiquetes | Relación <<include>> |
| Pagar tiquetes | Sistema de pago | Relación <<include>> |
| Reservar tiquetes | Recibir confirmación de compra | Relación <<include>> |
| Recibir confirmación | Sistema de notificación | Relación <<include>> |

## Vista Lógica

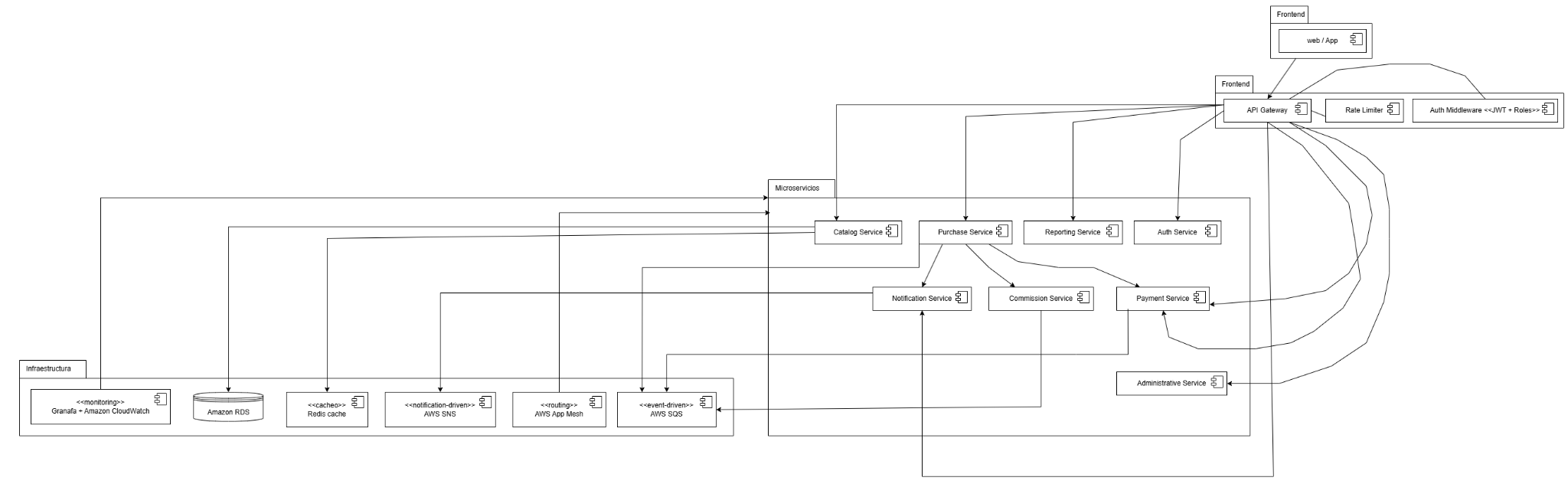


Este diagrama de actividades representa el flujo lógico del proceso de compra de un tiquete en el sistema Servidor de Tiquetes, desde que el revendedor realiza la solicitud hasta la confirmación y notificación del éxito. El flujo inicia con la autenticación mediante JWT a través del API Gateway. Si el token es válido, se reenvía la solicitud al Purchase Service, que consulta la disponibilidad del tiquete en el Catalog Service. Este servicio utiliza Redis Cache para una respuesta eficiente. Si el tiquete está disponible, el stock se bloquea temporalmente y se genera un evento "CompraIniciada" que es enviado a AWS SNS/SQS. Luego se solicita el pago al Payment Service. Si el pago es exitoso, se confirma la compra y se emite un nuevo evento "CompraCompletada". Este evento es procesado por servicios reactivos: Notification Service (envía notificación al usuario), Commission Service (calcula la comisión del revendedor), y Catalog Service (actualiza el estado del tiquete). Finalmente, el revendedor recibe la confirmación exitosa.

Tabla de elementos y relaciones

| **Actor** | **Elemento UML** | **Relación o Acción Principal** |
| --- | --- | --- |
| Revendedor | Inicio | Inicia el flujo de compra |
| API Gateway | Verifica JWT | Validación del token de autenticación con Auth Service |
| API Gateway | Reenvía solicitud | Si el JWT es válido, reenvía la solicitud a Purchase Service |
| Purchase Service | Solicita disponibilidad | Solicita estado del tiquete a Catalog |
| Catalog Service | Consulta Redis Cache | Optimiza la consulta de disponibilidad usando cache |
| Catalog Service | Responde disponibilidad | Informa si el tiquete está disponible |
| Purchase Service | Bloquea stock | Evita doble compra |
| Purchase Service | Emite evento "CompraIniciada" | Publica a AWS SNS/SQS para procesamiento asincrónico |
| Purchase Service | Solicita pago | Llama al Payment Service |
| Payment Service | Procesa pago con PSP | Interacción con proveedor de pagos |
| Payment Service | Responde a Purchase | Informa si el pago fue exitoso |
| Purchase Service | Confirma compra | Marca el tiquete como vendido |
| Purchase Service | Emite evento "CompraCompletada" | Publica a AWS SNS/SQS para disparar acciones reactivas |
| AWS SNS/SQS | Event dispatcher | Distribuye eventos a los microservicios reactivos |
| Notification Service | Notifica usuario | Informa al cliente que su compra fue exitosa |
| Commission Service | Calcula comisión | Asigna la comisión por la venta |
| Catalog Service | Actualiza estado del tiquete | Marca el tiquete como vendido en el catálogo |
| Revendedor | Recibe notificación | Se le informa que la compra fue completada correctamente |

## Vista Desarrollo

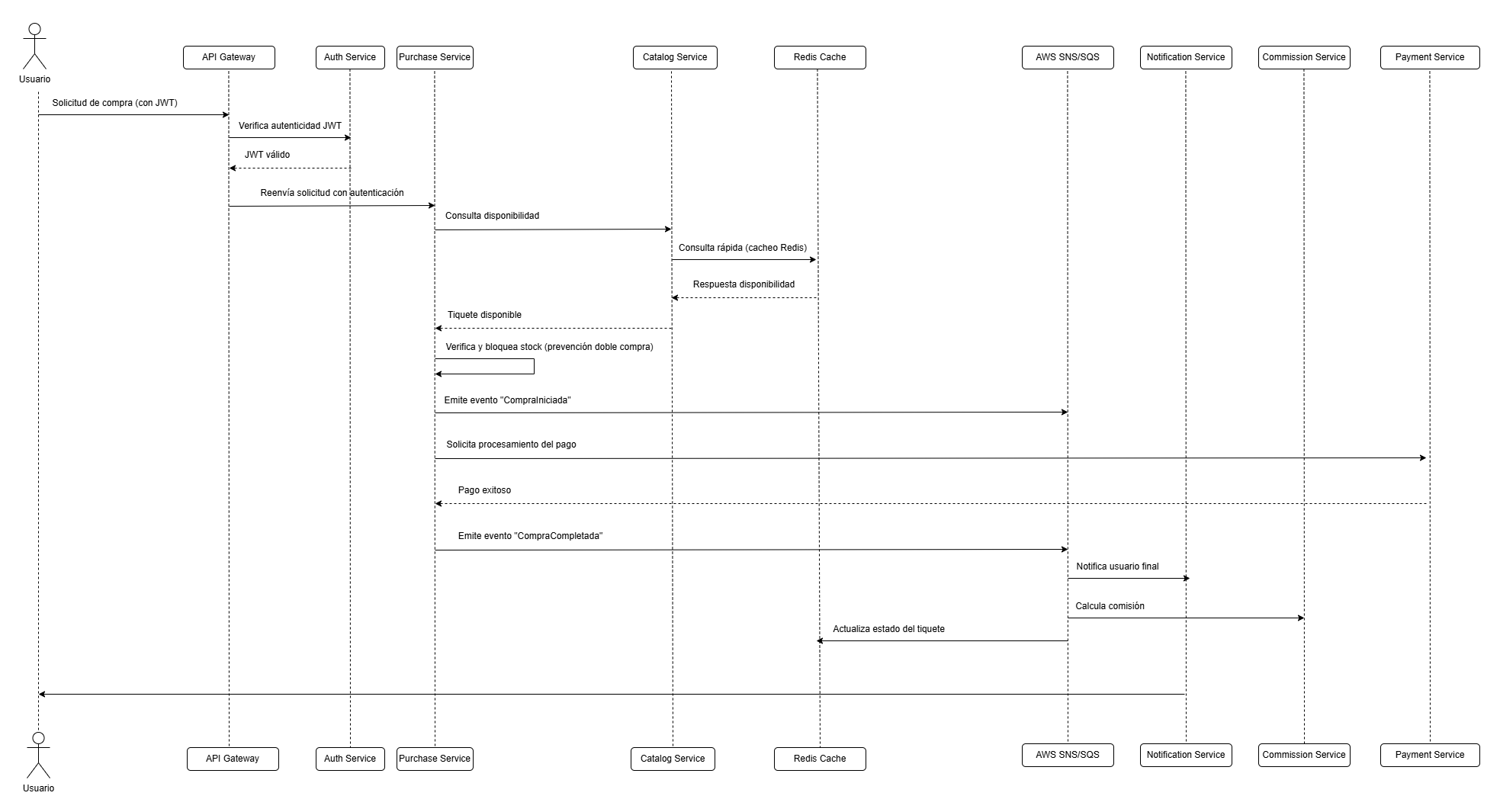


Este diagrama de componentes describe la arquitectura lógica del sistema Servidor de Tiquetes basado en microservicios, estructurado con una puerta de entrada central (API Gateway) y respaldado por una infraestructura cloud en AWS. Los clientes interactúan con la plataforma a través de una aplicación web o móvil que se conecta al API Gateway, el cual maneja la autenticación (JWT), autorización por roles y limitación de tráfico. El Gateway enruta las solicitudes a los diferentes microservicios responsables de funciones clave: Catálogo, Compras, Pagos, Notificaciones, Comisiones, Reportes, Administración y Autenticación. El sistema emplea tácticas de alto rendimiento como cacheo en Redis, mensajería asincrónica con AWS SQS/SNS, monitoreo con CloudWatch y Grafana, y enrutamiento inteligente con AWS App Mesh. Los datos persistentes son gestionados a través de Amazon RDS. Esta estructura desacoplada permite alta escalabilidad, trazabilidad y resiliencia ante fallos.

Tabla de elementos y relaciones

| **Actor / Sistema** | **Elemento UML** | **Relación** |
| --- | --- | --- |
| Usuario (cliente) | Web/App | Inicia solicitudes hacia el sistema |
| Web/App | API Gateway | Se comunica directamente para consumir funcionalidades |
| API Gateway | Auth Middleware | Maneja autenticación JWT y control de acceso por roles |
| API Gateway | Rate Limiter | Controla el tráfico por usuario o revendedor |
| API Gateway | Todos los microservicios | Enruta solicitudes hacia los microservicios específicos |
| Purchase Service | Notification Service | Emite notificaciones luego de compra |
| Purchase Service | Commission Service | Calcula la comisión correspondiente |
| Purchase Service | Payment Service | Gestiona el proceso de pago |
| Payment Service | Administrative Service | Informa eventos críticos o de fraude |
| Catalog Service | Redis Cache | Usa cacheo para mejorar la consulta de tiquetes |
| Catalog Service | Amazon RDS | Lee y escribe datos persistentes |
| Purchase Service | AWS SQS | Emite eventos de compra asincrónicos |
| Payment Service | AWS SQS | Emite eventos relacionados con pagos |
| Notification Service | AWS SNS | Publica notificaciones hacia clientes o sistemas externos |
| Commission Service | AWS SQS | Escucha eventos para procesar comisiones |
| Grafana + CloudWatch | Microservicios | Monitorea métricas, alertas y registros |
| AWS App Mesh | Microservicios | Orquesta llamadas internas seguras y observables |

## Vista Procesos

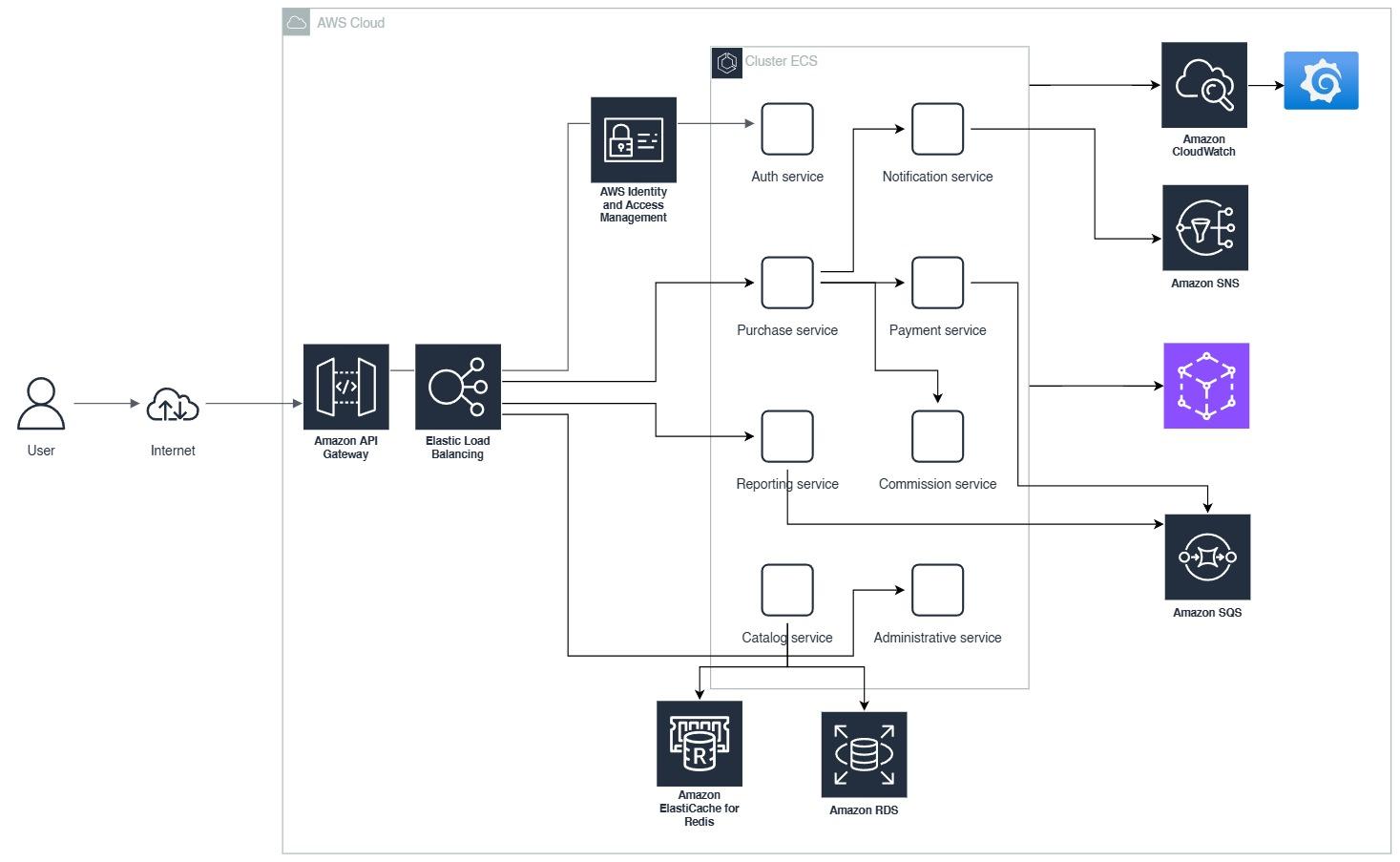


Este diagrama de secuencia representa el flujo completo de una compra de tiquete por parte de un revendedor, asegurando la autenticación segura, la validación de disponibilidad, la prevención de doble compra y la comunicación asincrónica de eventos a otros servicios. Inicia con la autenticación mediante JWT a través del API Gateway y el Auth Service. Luego, el Purchase Service verifica la disponibilidad del tiquete en el Catalog Service, que a su vez consulta Redis como caché. Si el tiquete está disponible, se bloquea temporalmente el stock y se emite un evento de "CompraIniciada". Posteriormente, el Payment Service procesa el pago y, si es exitoso, se confirma la compra y se emite un evento "CompraCompletada". Este evento es consumido por los servicios Notification, Commission y Catalog, que respectivamente notifican al usuario, calculan la comisión y actualizan el estado del tiquete. Finalmente, el revendedor recibe una notificación de éxito.

Tabla de elementos y relaciones

| **Actor** | **Elemento UML** | **Relación** |
| --- | --- | --- |
| Revendedor | API Gateway | Envia solicitud de compra con JWT |
| API Gateway | Auth Service | Verifica autenticidad del JWT |
| API Gateway | Purchase Service | Reenvía solicitud autenticada |
| Purchase | Catalog Service | Consulta disponibilidad del tiquete |
| Catalog | Redis Cache | Realiza cacheo para disponibilidad |
| Purchase | Purchase (interno) | Verifica y bloquea stock |
| Purchase | AWS SNS/SQS (Queue) | Emite evento "CompraIniciada" |
| Purchase | Payment Service | Solicita procesamiento de pago |
| Payment | Purchase Service | Retorna resultado del pago |
| Purchase | AWS SNS/SQS (Queue) | Emite evento "CompraCompletada" |
| SNS/SQS | Notification Service | Notifica al usuario final |
| SNS/SQS | Commission Service | Calcula comisión del revendedor |
| SNS/SQS | Catalog Service | Actualiza estado del tiquete |
| Notification | Revendedor | Envía notificación de éxito |

## Vista Física



Este diagrama representa la vista física del sistema Servidor de Tiquetes desplegado en AWS Cloud, aplicando una arquitectura basada en microservicios con alta disponibilidad, escalabilidad y separación de responsabilidades. Los usuarios acceden al sistema a través de Amazon API Gateway, que canaliza las solicitudes hacia un Elastic Load Balancer. Este balanceador distribuye el tráfico a los distintos microservicios desplegados en un ECS Cluster (con Fargate o EC2). Cada servicio tiene una función específica: autenticación (Auth), compras (Purchase), pagos (Payment), notificaciones, comisiones, reportes, administración y catálogo.

La infraestructura incluye Redis (ElasticCache) para mejorar el rendimiento en consultas de catálogo, y Amazon RDS para persistencia de datos.

Los eventos se manejan mediante Amazon SNS (para notificaciones), Amazon SQS (para colas de eventos asincrónicos) y todo el sistema es monitoreado con Amazon CloudWatch. El control de acceso está gestionado por AWS IAM.

Tabla de elementos y relaciones

| **Componente AWS** | **Tipo / Servicio** | **Relación / Función** |
| --- | --- | --- |
| User | Actor externo | Inicia la interacción a través de Internet |
| Internet | Medio | Conduce el tráfico al API Gateway |
| Amazon API Gateway | API Management | Entrada segura para solicitudes REST |
| Elastic Load Balancing (ALB) | Load Balancer | Distribuye solicitudes entre los microservicios del ECS Cluster |
| AWS IAM | Identidad y Seguridad | Verifica autenticación y permisos del usuario |
| ECS Cluster | Contenedor de microservicios | Alojamiento y ejecución de los servicios |
| Auth Service | Microservicio | Autenticación y autorización con JWT |
| Purchase Service | Microservicio | Gestión de compras y prevención de doble compra |
| Payment Service | Microservicio | Procesamiento de pagos |
| Notification Service | Microservicio | Envía notificaciones al usuario |
| Commission Service | Microservicio | Calcula y registra comisiones |
| Reporting Service | Microservicio | Genera informes operativos y financieros |
| Catalog Service | Microservicio | Consulta y actualización de disponibilidad de tiquetes |
| Administrative Service | Microservicio | Herramientas para soporte y administración manual |
| Amazon ElasticCache (Redis) | Caché distribuido | Cacheo de datos de catálogo para mejorar rendimiento |
| Amazon RDS | Base de datos relacional | Persistencia estructurada de información crítica |
| Amazon SNS | Servicio de notificaciones | Emite eventos de notificación en tiempo real |
| Amazon SQS | Servicio de colas asincrónicas | Manejo de eventos desacoplados entre servicios |
| Amazon CloudWatch | Monitoreo | Métricas, logs y alertas para observabilidad del sistema |